Characterising Polytime through higher type Recursion

Karl-Heinz Niggl

Characterising Polytime Through Higher Type Recursion

Karl–Heinz Niggl*
Mathematisches Institut der Ludwig–Maximilians–Universität München
Theresienstraße 39, 80333 München, Germany
e-mail: niggl@rz.mathematik.uni-muenchen.de

Abstract

It is shown how to restrict recursion on notation in all finite types so as to characterise the polynomial time computable functions. The restrictions are obtained by enriching the type structure with the formation of types $!\sigma$, and by adding linear concepts to the lambda calculus.

1 Introduction

Recursion in all finite types has been introduced by D. Hilbert [4]. For the purpose of consistency proofs, Hilbert extended his finitistic point of view and introduced a system supporting recursion in all finite types. This system later became known as Gödel's T [3].

According to [4], the value computed by a higher type recursion can be any functional, which is to say a mapping that takes other mappings as arguments and produces a new mapping. To make this precise, Hilbert introduced *types* built from the ground type ι for the natural numbers by means of the type operator $\to$, and he associated with each type σ the domain $\mathtt{HT}_\sigma$ of the hereditarily total functionals of type σ defined as follows: $\mathtt{HT}_\iota := \mathbb{N}$, and $\mathtt{HT}_{\sigma\to\tau}$ is the set of all mappings sending functionals in $\mathtt{HT}_\sigma$ to functionals in $\mathtt{HT}_\tau$. Accordingly, *recursion in type* σ is a mapping $\mathcal{R}_\sigma$ of type $\sigma \to (\iota \to \sigma \to \sigma) \to \iota \to \sigma$, taking three arguments and producing a functional of type σ. Adapted to recursion on notation, the computational meaning of this mapping is given by the following conversion rules:

$$\begin{aligned} \mathcal{R}_\sigma g h 0 &\mapsto g \\ \mathcal{R}_\sigma g h (s_i \mathbf{n}) &\mapsto h(s_i\mathbf{n})(\mathcal{R}_\sigma g h \mathbf{n}) \end{aligned}$$

where $s_i\mathbf{n}$ denotes a non-zero binary numeral. It is well-known that primitive recursion and recursion on notation in all finite types have the same expressive power, namely the ϵ_0-recursive functions. In this paper it is shown how to restrict recursion on notation in all finite types so as to characterise the polynomial time computable functions.

*The present paper evolved from work on higher type ramification in collaboration with S. Bellantoni and H. Schwichtenberg, and shall be integrated in a forthcoming joint publication.

Generalising Simmons' [6] and Bellantoni/Cook's [1] approach to all finite types, we will restrict the way in which the previously computed functional can be accessed in a recursion, which is to say that some input positions are or may be active (normal), while others are dormant (safe) in a not yet specified form. For this purpose, we enrich the type structure with the formation of $!\sigma$ for any type σ. Then the recursion position as well as the first input position of the step functional will have a type $!\iota$. And by requiring that σ is !-free we would like to express that the second input position of the step functional is safe. So the question is *What does this mean for higher type inputs?*.

The key observation is that by a single higher type recursion one can define proper Kalmar-elementary functions. The function e satisfying $|e(m)(n)| = 2^{|m|} + |n|$ can be defined by:

$$e := \mathcal{R}_{\iota\to\iota}\, s_1\, (\lambda u^{!\iota} V^{\iota\to\iota} y^{\iota}.V(Vy)).$$

The growth rate of this function is mirrored in the fact that the *previous function* V occurs nested, or in other words, V is used in a non-linear way. So if one wants characterise the polynomial time computable functions, then one has to rule out nested occurrences of the previous function in a recursion. This is the place where concepts of linearity come into play. So intuitively, objects of types $!\sigma$ can be used in a non-linear way, whereas objects of safe types can be used in a certain linear way only. Following the conversion rules above, the recursor $\mathcal{R}_\sigma$ will, therefore, be of type $\sigma \to !(!\iota \to \sigma \to \sigma) \to !\iota \to \sigma$ where σ is *safe*, i.e. !-free.

The paper will present a simply typed term system of so-called rna-terms similar to Gödel's T. The part "rna" stands for "relevance, necessitation, and affination", and it refers to the essential restrictions to application, introduction of !, and to lambda abstraction, which control the formation of rna-terms in order to obtain a system which is closed under reduction and whose expressive power coincides with the polynomial time computable functions. As for the critical inclusion, it is shown that for each closed rna-term t of number theoretical type one can find a polynomial q_t such that for all numerals $\vec{n}$, one can compute the normal form $nf(t\vec{n})$ in time $q_t(\sum |n_i|)$, where $|n_i|$ denotes the length of n_i. Thus, we arrive at a prototype of a simple functional programming language which supports recursion in all finite types, and which provides a polynomial time evaluation strategy for programs in the language.

2 Preliminaries

We start off with a system of general *terms* which can be considered a simple functional programming language where computation is normalisation.

Types are ι, and if σ and τ are types, then so are $\sigma \to \tau$ and $!\sigma$. The *ground types* are the types that do not contain "$\to$"; all other types are *higher types*. We assume that ! binds tighter than $\to$, and that $\to$ associates to the right. Thus, $\sigma \to !\tau \to \rho$ is $\sigma \to (!\tau \to \rho)$. One writes $\vec{\sigma} \to \tau$ for $\sigma_1 \to \cdots \to \sigma_n \to \tau$; if $\vec{\sigma}$ is empty, then $\vec{\sigma} \to \tau$ is just τ.

Types fall into two groups: *complete* types of the form $\vec{\sigma} \to !\tau$, and *incomplete* types of the form $\vec{\sigma} \to \iota$. *Safe* types are !-free types.

Definition 1 Terms *and their types denoted* $r^\sigma, s^\sigma, t^\sigma$ *are defined by the following rules:*

(V) *Every variable* x^σ *is a term of type* σ.

(C) *Every constant symbol* $0^\iota, s_0^{\iota\to\iota}, s_1^{\iota\to\iota}, p^{\iota\to\iota}$ *is a term of indicated type.*

($!\,In$) *If* r^σ *is a term, then* $!\,r$ *is a term of type* $!\sigma$.

($!\,E$) *If* $r^{!\sigma}$ *is a term, then* κr *is a term of type* σ.

(App) *If* $r^{\sigma\to\tau}$ *and* s^σ *are terms, then* rs *is a term of type* τ.

(λ) *If* r^τ *is a term, then* $\lambda x^\sigma.r$ *is a term of type* $\sigma\to\tau$.

(D) *If* t_1 *is a term of type* ι, *and* t_2, t_3, t_4 *are terms all of type* $\iota\to\sigma$ *where* σ *is safe, then* $d_\sigma t_1 t_2 t_3 t_4$ *is a term of type* σ.

(R) *If* σ *is a safe type and* g, h, n *are terms of types* $\sigma, !(!\iota\to\sigma\to\sigma), !\iota$ *respectively, then* $\mathcal{R}_\sigma ghn$ *is a term of type* σ.

The constants s_0, s_1 stand for the usual *binary successors*, and p for the *binary predecessor*. d_σ is called *conditional* and accounts for *case distinction in the type* σ. Finally, $\mathcal{R}_\sigma$ is called *recursor* and accounts for *safe recursion on notation in type* σ.

We sometimes add parenthesis for clarity. Iterated application of terms like rst is understood as $(rs)t$. Since the symbols $!, \kappa$ are not constants, a term of the form $r\,!\,s\,t$ reads as $r(!\,s)t$.

Similar to Gödel's T, terms are interpreted over the hereditarily total functionals HT_σ of type σ, where $\mathrm{HT}_{!\sigma} := \mathrm{HT}_\sigma$. Thus, in the semantics we identify objects of type $!\sigma$ with those of type σ, since we are only interested in the computational behaviour of terms. As usual one defines the *value* $[\![t]\!]_\varphi$ *of a term* t *in an environment* φ, where $[\![!\,r]\!]_\varphi := [\![r]\!]_\varphi$ and $[\![\kappa r]\!]_\varphi := [\![r]\!]_\varphi$. As the value of a closed term t is independent of any environment, we just write $[\![t]\!]$.

In order to compute the value $[\![t]\!]$ of closed terms t of ground type, we introduce an *operational semantics* for terms given as usual by a *reduction relation* $r \longrightarrow r'$ between terms r, r'. Now "$\longrightarrow$" is induced by the *conversion rules* $r \mapsto r'$ listed below. In order to give consistent conversion rules for d_σ and $\mathcal{R}_\sigma$ redexes, we refer to the notion of binary numeral. A *binary numeral* is either 0, or else it is of the form $s_{i_1} \dots s_{i_k} s_1 0$ where $i_j \in \{0, 1\}$. Binary numerals distinct from 0 are denoted by $s_i\mathbf{n}$. So the conversion rules are as follows:

$$
\begin{array}{rl}
s_0 0 \mapsto 0 & d_\sigma 0 t_2 t_3 t_4 \mapsto t_2 0 \\
p0 \mapsto 0 & d_\sigma (s_i\mathbf{n}) t_2 t_3 t_4 \mapsto t_{i+3}(s_i\mathbf{n}) \\
p(s_i t) \mapsto t & \kappa(!\,t) \mapsto t \\
(\lambda x.r)s \mapsto r[s/x] & \mathcal{R}_\sigma gh\,!0 \mapsto g \\
 & \mathcal{R}_\sigma gh\,!(s_i\mathbf{n}) \mapsto (\kappa h)\,!(s_i\mathbf{n})(\mathcal{R}_\sigma gh\,!\mathbf{n})
\end{array}
$$

A *redex* is the left-hand side term of a conversion rule. As usual $(\lambda x.r)s$ is called a β *redex* and $(\lambda x.r)s \mapsto r[s/x]$ is a β *conversion*. All other redexes and conversion rules are named after the leading left-hand side symbol, e.g. $\kappa(!\,t)$ is a κ redex.

Note that the restriction to binary numerals in case of d_σ and $\mathcal{R}_\sigma$ redexes does not restrict the expressive power of terms but just the freedom of converting redexes in arbitrary order.

Let "$\longrightarrow^*$" denote the reflexive and transitive closure of "$\longrightarrow$". We say that t' is a *reduct* of t if $t \longrightarrow^* t'$. A *reduction sequence for* t is a maximal sequence $t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots$ of reductions starting with t. A term is *normal* if it has no redexes. A *numeral* is of the form $!\ldots!\,\mathbf{n}$ where $\mathbf{n}$ is a binary numeral.

Theorem 1 (Folklore) *Every reduction sequence for a term t terminates in a unique normal term $nf(t)$. In particular, if t is a closed and ground, then $nf(t)$ is a numeral denoting $[\![t]\!]$.*

Note. Normal terms are of the form $\lambda\vec{x}.U\vec{r}$ where $\vec{r}$ are normal, and U is either a variable or a constant $0, s_0, s_1, p$, or else a symbol $!, \kappa, d_\sigma, \mathcal{R}_\sigma$.

By theorem 1 the system of terms can be considered a simple functional programming language where computation is normalisation. While preserving this computational aspect, we will restrict the formation of terms so as to characterise the polynomial time computable functions.

A term t^σ is said to be *complete*, *incomplete*, *safe*, etc. if σ is complete, incomplete, safe, etc.

Notions like *subterm* of and *free variable* or *bound variable* in a term are as usual. For terms r, s we sometimes write $r \subseteq s$ to stand for r *is a subterm of* s, and $FV(r)$ to denote the set of free variables of r, whereas $Var(r)$ is used to denote the set of variables of r, bound or free. Furthermore, we write $FO(x, r)$ to denote the multi-set of *free occurrences of* x *in* r.

The *length* $|t|$ of a term t is defined by $|x| := |c| := 1$ for constants c, $|Ur| := 1 + |r|$ for $U = !, \kappa$, $|rs| := 1 + |r| + |s|$, $|\lambda x.r| := 1 + |r|$, and $|d_\sigma t_1 t_2 t_3 t_4| := 1 + \sum |t_i|$.

We identify α-*equal* terms, i.e. terms which differ only by a renaming of bound variables. For terms $r, s_1, \ldots, s_k$ and distinct variables $x_1, \ldots, x_k$ with x_i being of the same type as s_i, one defines as usual what it means that $s_1, \ldots, s_k$ *are simultaneously substitutable for* $x_1, \ldots, x_k$ *in* r, and in that case we denote by $r[\vec{s}/\vec{x}]$ the term obtained by simultaneously substituting each s_i for the free occurrences of x_i in r. Identifying α-equal terms, we can always assume that $\vec{s}$ are simultaneously substitutable for $\vec{x}$ in r when writing $r[\vec{s}/\vec{x}]$.

By a *use* of a variable x in a term t we mean an occurrence of x in t other than λx. If there is only one such occurrence of x in t, then we speak of a *linear* use of x in t.

Occurrences $\vec{r}'$ of subterms $\vec{r}$ in a term r are said to be *scope equivalent in* r if all $\vec{r}'$ occur in s whenever $\lambda y.s \subseteq r$ such that some r_i' occurs in s with $y \in FV(r_i)$.

3 RNA-Terms

In this section we will define a subclass of terms called rna-terms. It is shown that rna-terms have the expressive power of the polynomial time computable functions.

The design of rna-terms is motivated by the desire to have a subclass of terms which is closed under reduction and thus can be viewed as a simple functional programming language, and for which it is decidable whether or not a given arbitrary term belongs to that subclass.

Definition 2 *For terms r and incomplete variables x, we say:*

(*i*) a^{ι} is an affination of x in r *if* $a \subseteq r$ *and* $|FO(x,a)| = 1$, *every free occurrence of x in r is in an occurrence of a in r, and the occurrences of a in r are scope equivalent in r.*

(*ii*) x is affinable in r *if there is an affination of x in r, or* $|FO(x,r)| \leq 1$.

Definition 3 RNA-Terms *are defined by* (V), (C), $(!\,E)$, (D), (R), *and the modified rules:*

$(!\,I)$ *If r^{σ} is an rna-term with $FV(r)$ all complete, then $!\,r$ is an rna-term of type $!\sigma$.*

(A) *If $r^{\sigma\to\tau}$ and s^{σ} are rna-terms, then rs is a rna-term of type τ; provided that if r is complete, then $FV(s)$ are all complete.*

(L) *If r^{τ} is an rna-term, then $\lambda x^{\sigma}.r$ is an rna-term of type $\sigma \to \tau$; provided that if σ is incomplete, then x is affinable in a reduct of r.*

Note. Every subterm of an rna-term is an rna-term. Furthermore, it is decidable whether a given term is an rna-term. Finally, observe that in (L) there is no restriction on lambda abstracted variables x of type ι, for the free occurrences of x in r form an affination of x in r.

In the notion of rna-terms, "r" stands for *relevance* explained below, "n" for *necessitation* due to $(!\,I)$, and "a" stands for *affinability*. The rule $(!\,I)$ is a simplification of Prawitz's ! introduction rule* in his formulation of S4. The simplification is due to the presence of the rule (A).

As pointed out in the introduction, the intuition behind objects of complete types is that they can be used in a non-linear way, while objects of incomplete types can be used in a certain linear way only, expressed by the notion of affinability. Therefore, when introducing a term $!\,r$, it is consistent to require that r has no incomplete free variables. Accordingly, the restriction on applications expressed in (A) rules out that objects of incomplete types can be imported by application and reduction into objects of multiple usage. This goes hand in hand

*Here we are indebted to Grigori Mints for reminding us of Prawitz's rule, and for pointing out that the system of general terms based on Prawitz's rule is closed under reduction.

with the desire that the concept of restricted usage of bound incomplete variables expressed in (L) is preserved under reduction. Consider for example the following conversion:

$$\lambda x^{\sigma} \ldots (\lambda y^{!\tau}.r)(Y^{\sigma \to !\tau} x) \ldots \quad \mapsto \quad \lambda x^{\sigma} \ldots r[Y^{\sigma \to !\tau} x/y] \ldots$$

Suppose that the use of x^{σ} in the left-hand side term is linear, and assume that $!\tau$ is also higher-type. As $y^{!\tau}$ is allowed to have multiple usage in s, the use of x in the right-hand side term might be neither linear nor affinable.

Although higher-type complete variables cannot occur in normal terms defining number-theoretical functions, they can show up in non-normal terms. These kind of detours make life very hard when trying to establish properties of terms which should be preserved under reductions, such as affinability. It is entirely due to the rules (A) and $(!I)$ that the free variables of complete terms are always complete. In that way these two rules account for the fact that every use of an incomplete variable in an rna-term t is *relevant* in the sense that it cannot be imported through a higher-type complete variable which disappears in the normal form of t.

As for (L), the intuition of restricted usage of bound incomplete variables is essentially expressed in the notion of affinability. But this does not lead to a system which is closed under reduction. Consider for example an rna-term r and an incomplete variable x such that x is affinable in r. Furthermore, suppose that x has an affination a in r. Now consider a term r' obtained from r by reducing one occurrence of a in r to a', that is

$$r = \ldots a \ldots a \ldots a \ldots \longrightarrow \ldots a \ldots a' \ldots a \ldots = r'.$$

If x has still a free occurrence in a', then neither x has at most one free occurrence in r' nor does x have an affination in r', which is to say that x is not affinable in r'. Thus, if (L) would only require that x is affinable in r, then $\lambda x.r$ would be an rna-term, but not $\lambda x.r'$. So the idea behind requiring in (L) that x is affinable in a reduct of r is that every reduction locally performed in an occurrence of a in r can be successively performed in all occurrences of a in r, leading to a term $r'' = \ldots a' \ldots a' \ldots a' \ldots$ where now a' is an affination of x in the reduct r'' of r. In fact, this idea leads to a system which is closed under reduction.

It remains to add that dropping in definition 2 the requirement on affinations to be scope equivalent would allow one to define the Kalmar-elementary function e above by

$$e := \mathcal{R}_{\iota \to \iota}\, s_1\, (\lambda u^{!\iota} V^{\iota \to \iota} y^{\iota}.(\lambda y^{\iota}.Vy)(Vy)).$$

For then the two occurrences of Vy would give an affination of V in e, and thus e would be an rna-term. Thus, scope equivalence rules out the hiding $(\lambda y^{\iota}.Vy)(Vy)$ of nested occurrences of the *previous function* V in a recursion.

4 Closure Properties of RNA-terms

The aim of this section is to show that rna-terms are closed under reductions. To this end, we first establish some other properties of rna-terms.

Lemma 1 (Relevance) *Every complete rna-term r has complete free variables only.*

Proof. We proceed by induction on the structure of complete rna-terms r. For type reasons, r is neither a constant nor a symbol $d_\sigma, \mathcal{R}_\sigma$. If r is a variable, then the statement of the lemma is obvious. If r is $!\,s$, then s has complete free variables only, and so does r. If r is κs, then s is complete, and we are done by the induction hypothesis on s. If r is $s^{\sigma\to\tau}t^\sigma$, then s is complete and t contains complete free variables only, since r is complete. Hence we are done by the induction hypothesis on s. Finally, if the complete r is $\lambda x.s$, then s is complete, and we are done by the induction hypothesis on s. □

Lemma 2 (Closure under Substitution) *If $r, \vec{s}$ are rna-terms, then so is $r[\vec{s}/\vec{x}]$.*

Proof. We proceed by induction on the structure of r. If r is a variable or a constant, then $r[\vec{s}/\vec{x}]$ is either some s_i, or else it is r; thus we are done.

If r is $!\,s$, then s has complete free variables only. Now $s[\vec{s}/\vec{x}]$ is an rna-term by the induction hypothesis, and lemma 1 implies that each s_i being substituted for a free occurrence of x_i in s has complete free variables only. Hence $r[\vec{s}/\vec{x}] = !\,s[\vec{s}/\vec{x}]$ is an rna-term.

If r is κs, then $r[\vec{s}/\vec{x}] = \kappa s[\vec{s}/\vec{x}]$ is an rna-term by the induction hypothesis on s.

Suppose that r is $s^{\sigma\to\tau}t^\sigma$. Hence $r[\vec{s}/\vec{x}] = (s[\vec{s}/\vec{x}])(t[\vec{s}/\vec{x}])$, and $s[\vec{s}/\vec{x}], t[\vec{s}/\vec{x}]$ are rna-terms by the induction hypothesis. If s is incomplete, then we are done. Otherwise s is complete, and t has complete free variables only. Hence every s_i being substituted for a free occurrence of x_i in t is complete. Therefore lemma 1 implies that $t[\vec{s}/\vec{x}]$ has complete free variables only, and so $r[\vec{s}/\vec{x}]$ is an rna-term.

Suppose that r is $r = \lambda x.s$. We may assume $x \notin \vec{x}$, and $x \notin FV(s_i)$ for all s_i. Then we have $r[\vec{s}/\vec{x}] = \lambda x.s[\vec{s}/\vec{x}]$, and $s[\vec{s}/\vec{x}]$ is an rna-term by the induction hypothesis. If x is complete, then we are done. Otherwise x is incomplete, and x is affinable in a reduct r'' of r'. Hence x is affinable in the reduct $r''[\vec{s}/\vec{x}]$ of $r'[\vec{s}/\vec{x}]$, and $r[\vec{s}/\vec{x}]$ is an rna-term.

If r is $U\vec{r}$ where U is d_σ or $\mathcal{R}_\sigma$, then we are done using the induction hypothesis on $\vec{r}$. □

Lemma 3 (Affinability) *Let r be any rna-term such that x is affinable in r. If $r \longrightarrow r'$, then x is affinable in a reduct of r'.*

Proof. We proceed by induction on the structure of r, assuming $r \longrightarrow r'$. We may assume $|FO(x,r)| \geq 1$. First consider the case $|FO(x,r)| = 1$. We may assume $|FO(x,r')| \geq 2$, and by lemma 1 the only conversion rule capable of multiplying an incomplete variable is a β conversion. Thus, r' results from r by converting a redex $(\lambda y.s)t$ where the unique free occurrence of x in r is in t. Since x has incomplete type, lemma 1 gives that t has also incomplete type, and so does y. Since $|FO(x,r')| \geq 2$, this implies that y has an affination b in s. Hence $b[t/y]$ is an affination of x in r', and we are done.

Suppose that $|FO(x,r)| \geq 2$. Hence x has an affination a in r. If the reduction $r \longrightarrow r'$ doesn't involve a, then a is still an affination of x in r'. Otherwise either r' results from r by reducing an occurrence of a to a', or some occurrences of a in r vanish in r', in which case we are done, or some occurrences of a in r are multiplied in r', in which case an argument similar to that above shows that a is still an affination of x in r', or else r' results from r by converting a redex $(\lambda y.s)t$ where a has an occurrence in s such that $y \in FV(a)$. In the latter case, by scope equivalence all occurrences of a in r must be in s, implying that $a[t/y]$ is an affination of x in r'. In the former case, the induction hypothesis on a gives that x is affinable in a reduct a'' of a' – since a is a proper subterm of r, and x is affinable in a. Hence x is affinable in the reduct r'' of r' obtained from r by successively reducing all occurrences of a in r to a''. □

Lemma 4 *Let r be any rna-term such that every reduct of r is also an rna-term, and such that x is affinable in r. Then x is affinable in $nf(r)$.*

Proof. By the folklore theorem, we proceed by induction on the height $h(r)$ of the reduction tree of r. The *base case* $h(r) = 0$ is obvious. As for the *step case* $h(r) > 0$, we have $r \longrightarrow r'$ for some r'. Now lemma 3 implies that x is affinable in a reduct r'' of r', and by the hypothesis on r, r'' is an rna-term such that all reducts of r'' are rna-terms, too. Therefore the induction hypothesis on r'' gives that x is affinable in $nf(r'') = nf(r)$. □

Theorem 2 (Closure under Reduction) *If $r \longrightarrow^* r'$ and r is an rna-term, then so is r'.*

Proof. We proceed by induction on the height $h(r)$ of the reduction tree of r, and by side induction on the structure of r. Assume that $r \longrightarrow r_1 \longrightarrow^* r'$. By the induction hypothesis on r_1, it suffices to show that r_1 is an rna-term. The case that r is a variable or a constant is ruled out, since that r is not reducible.

If r is $!\,s$, then s has complete free variables only, and r_1 is $!\,s_1$ with $s \longrightarrow s_1$. Now s_1 is an rna-term by the side induction hypothesis on s. Hence r_1 is an rna-term, as $FV(s_1) \subseteq FV(s)$.

If r is κs, then either s is $!\,s_1$ and r_1 is s_1, in which case we are done, or else r_1 is κs_1 and $s \longrightarrow s_1$. In the latter case, r_1 is an rna-term by the side induction hypothesis on s.

Suppose that r is $s^{\sigma\to\tau}t^{\sigma}$. If r_1 results from r by reducing either s or t, then r_1 is an rna-term by the side induction hypothesis on the reduced subterm. Otherwise st is itself a redex, and r_1 results from r by converting that redex. Thus, it suffices to show that if the left-hand side term of a conversion rule is an rna-term, then so is the right-hand side term. In case of a β conversion, this follows from lemma 2. The claim is obvious for a successor or a predecessor or a κ conversion. In case of a d conversion, e.g. $d_\sigma(s_1\mathbf{n})t_2t_3t_4 \mapsto t_4(s_1\mathbf{n})$, observe that t_4 and $s_1\mathbf{n}$ are rna-terms of safe types. Hence $t_4(s_1\mathbf{n})$ is an rna-term formed by (A). Finally, in case of an $\mathcal{R}$ conversion, e.g. $\mathcal{R}_\sigma gh\,!\,s_i\mathbf{n} \mapsto (\kappa h)\,!(s_i\mathbf{n})(\mathcal{R}_\sigma gh\,!\mathbf{n})$, observe that $!\mathbf{n}$ is an rna-term, and that κh is an incomplete rna-term. Hence by (A) the right-hand side term is an rna-term.

Suppose that r is $\lambda x.s$. Hence $r_1 = \lambda x.s_1$ and $s \longrightarrow s'$. We may assume that is x incomplete. Hence x is affinable in a reduct s' of s, and we find ourselves in the situation:

$$\begin{array}{c} s \longrightarrow^* s' \\ \downarrow \\ s_1 \end{array}$$

Now the side induction hypothesis gives that s_1 and s' are rna-terms, and that all reducts of s' are rna-terms, too. Therefore lemma 4 implies that x is affinable in $nf(s') = nf(s) = nf(s_1)$, showing that r_1 is an rna-term.

Finally, suppose that r is $U\vec{t}$ where U is d_σ or $\mathcal{R}_\sigma$. Then either r_1 results from r by reducing one of the subterms $\vec{t}$, in which case we are done using the side induction hypothesis, or else r_1 results from r by converting the redex r. The latter case is already treated above. □

The folklore theorem and the closure of rna-terms under reduction implies the corollary below.

Theorem 3 (Strong Normalisation) *Every reduction sequence for an rna-term r terminates in a unique normal rna-term $nf(r)$. In particular, if r is closed and ground, then $nf(r)$ is a numeral denoting $[\![r]\!]$.* □

5 Analysis of Normal Terms

The purpose of this section is to analyse the structure of normal terms, exploring the effect of the type restrictions with respect to those terms representing number theoretical functions. Lemma 6 below implies that those terms use safe or ground type variables only. Thus, in our analysis we may restrict to terms using safe or ground type variables only.

Lemma 5 (Kappa) *If κr is a normal term, then $r = \kappa(\kappa \cdots \kappa(X\vec{r}_1)\vec{r}_2)\ldots)\vec{r}_n$ for some complete variable X.*

Proof. Induction on the structure of normal terms r. For type reasons, r is of the form $U\vec{s}$ where U is neither a constant nor a symbol $d_\sigma, \mathcal{R}_\sigma$. Since κr is normal, U is not the symbol !. If U is a variable X, then it has a type $\vec{\sigma} \to !\tau$ and we are done. If U is κ, then $\#\vec{s} \geq 1$ and the induction hypothesis for s_1 gives a term of the required form, hence $\kappa\vec{s} = r$ has the required form. □

Definition 4 *For types σ, ρ we say σ* occurs strictly left in *ρ, denoted by $\sigma \subset_l \rho$, iff either $\rho = \rho_1 \to \rho_2$ and ($\sigma \subseteq_l \rho_1$ or $\sigma \subset_l \rho_2$), or else $\rho = !\rho'$ and $\sigma \subset_l \rho'$. Here $\subseteq_l$ means $\subset_l$ or $=$.*

Lemma 6 *If t^σ is a normal term and x^τ is a higher-type non-safe variable of t, then $(*)$ $\tau \subset_l \sigma$ or else $\tau \subseteq_l \rho$ for some $y^\rho \in FV(t)$.*

Proof. We proceed by induction on the structure of normal terms t of any type σ.

Suppose that t is $\lambda z^{\sigma_1}.r^{\sigma_2}$. If $x = z$ then $\tau = \sigma_1 \subset_l \sigma_1 \to \sigma_2 = \sigma$. Otherwise the induction hypothesis gives either $\tau \subset_l \sigma_2$ and so $\tau \subset_l \sigma$ as required, or else $\tau \subseteq_l \rho$ for some $y^\rho \in FV(r)$. In the latter case, if $y \neq z$ then $y \in FV(t)$; otherwise $y = z$ and $\tau \subset_l \sigma_1$, hence $\tau \subset_l \sigma$.

Suppose that t is $X\vec{r}$. If $\#\vec{r} = 0$ or $X = x$, then the second alternative of $(*)$ holds. Otherwise x appears in some $r_i^{\sigma_i}$ and the induction hypothesis gives either $\tau \subset_l \sigma_i$, implying that τ occurs strictly left in the type of $X \in FV(t)$, or else $\tau \subseteq_l \rho$ for some $y^\rho \in FV(r_i) \subseteq FV(t)$.

Suppose that t is $c\vec{r}$ where c is either a constant or a symbol $d_\sigma, \mathcal{R}_\sigma$. Then either x occurs in some safe or ground type subterm, or else c is $\mathcal{R}$ and x occurs in the step term r_2 of type $!(!\iota \to \rho)$ for some safe ρ. In either case, the induction hypothesis gives the second alternative of $(*)$. This is obvious in all cases, except possibly where x appears in r_2. Here the assumption $\tau \subset_l !(!\iota \to \rho)$ would imply $\tau \subset_l !\iota \to \rho$, contradicting that τ is higher-type and non-safe.

If t is $!\, r^\sigma$, then the induction hypothesis gives either $\tau \subset_l \sigma$, implying $\tau \subset_l !\sigma$ as required, or else $\tau \subseteq_l \rho$ for some $y^\rho \in FV(r) = FV(t)$.

Now let t be $\kappa\vec{r}$. Then $\#\vec{r} \geq 1$ and lemma 5 yields $t = \kappa(\ldots\kappa(X\vec{r}_1)\vec{r}_2)\ldots)\vec{r}_n$ for an complete variable X. If X is x, then the second alternative of $(*)$ holds. Otherwise x appears in some r_{ij} of type σ_{ij}. The induction hypothesis gives either $\tau \subset_l \sigma_{ij}$, implying that τ occurs strictly left in the type of $X \in FV(t)$, or else $\tau \subseteq_l \rho$ for some $y^\rho \in FV(r_{ij}) \subseteq FV(t)$. □

Corollary 1 *A closed normal term t of type $\sigma = \vec{\sigma} \to \tau$ with $\vec{\sigma}, \tau$ all safe or ground, contains only safe or ground type variables.* □

A term t is called *simple* if it contains safe or ground type variables only, and t has safe or ground type, or else type $!\iota \to \sigma$ for some safe σ.

Note. Let t be a simple normal rna-term. Then all subterms κr of t are of the form $\vec{\kappa}y$ for some complete ground type variable y. Furthermore, if rs is a subterm of t, then r has safe type. Finally, if $\lambda x.r$ is a subterm of t, then either x is affinable in r, or else x is of complete ground type.

6 The Bounding Lemma

In this section we will show that the number theoretical functions definable in the system of rna-terms are all polynomial time computable functions. Essentially this follows from the Bounding lemma below. It states that for every closed rna-term t of number theoretical type $\vec{\tau} \to \tau$, that is, all $\vec{\tau}, \tau$ are ground, one can find a polynomial q_t such that for every suitable list of numerals $\vec{n}$ one can compute the numeral $nf(t\vec{n})$ in time $q_t(\sum |n_i|)$.

For this purpose we define a suitable subset of rna-terms called *rna-terms with sharing*, or *srna-terms* for short. Using srna-terms one can show that normalisation of rna-terms doesn't lead to the usual explosion of complexity with respect to length and time. This is due to an

explicit control over the substitutions to be performed. In that way, srna-terms provide the control structure needed when computing $nf(t\vec{n})$ in terms of the given inputs $\vec{n}$, i.e. when bounding the time needed to compute $nf(t\vec{n})$ by a polynomial in the lengths of the inputs $\vec{n}$.

Definition 5 SRNA-Terms *are rna-terms r satisfying*

(VS) *every variable of r has safe or ground type,*

(AS) *all subterms st are of safe type,*

(SH) *if $\lambda x.s$ is a subterm of r, then either $|FO(x,s)| \leq 1$, or $\lambda x.s$ occurs as $(\lambda x.s)t$ where x is ground,*

(K) *all subterms κs are of ground type.*

Note. Every srna-term is of the form $U\vec{r}$ where U is either $\lambda x.s$ with $|FO(x,s)| \leq 1$, or U is a variable or a symbol $0, s_0, s_1, p, !, \kappa, d, \mathcal{R}$, or else U is $(\lambda x.s)t$ where x is of ground type.

Note. Every srna-term of higher-type $!\,\sigma$ is of the form $!\,r$, and every srna-term of type $!\iota \to \sigma$ is of the form $\lambda x^{!\iota}.r$. Furthermore, srna-terms are closed under non-$\mathcal{R}$ reductions.

Lemma 7 (Embedding of rna-terms) *Every simple normal rna-term t has an* srna-representation $\alpha(t)$ *such that $\alpha(t) \longrightarrow^* t$.*

Proof. Extending affinability, an affination of a complete ground type variable x in r consists of the free occurrences of x in r. Let t be any simple normal rna-term t. Then $\alpha(t)$ is obtained in at most $|t|$ transformation steps $t := t_0 \leftarrow t_1 \leftarrow \ldots \leftarrow \alpha(t)$ where each t_i has subterms rs of safe types only. Given t_i, we search in it for the leftmost not yet considered subterm $\lambda x.r$ with $|FO(x,r)| \geq 2$. Then x has an affination a in r, and r has a minimal safe subterm r' such that r' contains occurrences of a in r. Now let t_{i+1} result from t_i by replacing r' with $(\lambda y.r'[y/a])a$ for some new variable y.

First we show that every affination in t_i right from $\lambda x.r$ results in an affination in t_{i+1}. To see this, let $\lambda z.s$ be a subterm of r such that z has an affination b in s. If a has no occurrence in s, then b is still an affination of z in t_{i+1}. Otherwise by scope equivalence, either all occurrences of a are in s and $z \in FV(a)$, or else no occurrence of a in s has a free occurrence of z. In the latter case, either a occurs in b, in which case $b[y/a]$ is an affination of z in t_{i+1}, or else a, b are separated, in which case b is still an affination of z in t_{i+1}. In the former case, the minimality of r' implies that r' is in s. By construction t_{i+1} results from t_i by replacing the subterm r' of s with $(\lambda y.r'[y/a])a$ for some new y. Since z has a free occurrence in both a and b, there are two cases. If a is a subterm of b, then each occurrence of b contains exactly one occurrence of a, for b is an affination of z in s. By construction it follows that a is an affination of z in t_{i+1}. Otherwise if b is a subterm of a, then by construction b is still an affination of z in t_{i+1}.

It remains to show that the structure of every subterm $\lambda z.s$ of t_i where s has a subterm $(\lambda y'.s')b$ satisfying $FO(z,s) = FO(z,b)$ and $|FO(z,s)| = 1$, is preserved in t_{i+1}. To see this,

assume that $\lambda x.r$ occurs in s. If a, b are separated, then b occurs unchanged in t_{i+1}. If a occurs in b, then $(\lambda y'.s')b[y/a]$ occurs in t_{i+1}. If b would occur in a, then $|FO(z,s)| = |FO(x,r)| \geq 2$, contradicting $|FO(z,s)| = 1$. □

In the following, by *step* we mean a *computation step* with respect to any reasonable computation model (programming language) where the algorithms described (essentially in the lemmata 8, 9, 10) can be carried out. Roughly put, the overall strategy to prove the Bounding lemma is that we first show that one can bound the number of steps needed to compute $nf(t\vec{n})$ by $q_t(\sum |n_i|)$ for a suitable polynomial q_t. The real *time* needed to compute $nf(t\vec{n})$ is then obtained by giving a uniform *weight* $q_{time}(|s|)$ on each step performed on every term s occurring in the computation. For this strategy it is required that the length $|s|$ of every subterm occurring in the computation is also bounded by $q_t(\sum |n_i|)$.

Definition 6 *An srna-term is called* prenormal *if the only redexes in t are $\mathcal{R}$-redexes or ground type β redexes $(\lambda x.s)t$, that is, x is ground.*

Lemma 8 (Prenormal form) *For every srna-term t one can compute a prenormal srna-term $pnf(t)$ in at most $|t|$ steps such that $t \longrightarrow^* pnf(t)$, and every subterm occurring in the computation satisfies $|s| \leq |t|$.*

Proof. Given an srna-term t, the reduction strategy is that we convert all non-$\mathcal{R}$ redexes except those ground type β redexes $(\lambda x.s)t$ with $|FO(x,s)| \geq 2$. In doing so, every such step strictly decreases the length of the resulting term. Therefore, every reduction sequence for t based on that strategy must terminate in at most $|t|$ steps in a unique prenormal srna-term $pnf(t)$, and every subterm occurring in the computation satisfies $|s| \leq |t|$. □

Using the previous lemma, the next lemma prepares an estimation of the number of steps needed to compute the normal form of $\mathcal{R}$-free closed srna-terms of ground type.

Lemma 9 *Let t be any $\mathcal{R}$-free prenormal srna-term of ground type, and let $\vec{n}$ be any list of numerals such that $t[\vec{n}/\vec{x}]$ is closed. Then*

(*i*) *one can compute $nf(t[\vec{n}/\vec{x}])$ in at most $|t|^2$ steps, and*

(*ii*) *every term s occuring in the computation satisfies $|s| \leq |t| + \max |n_i|$.*

Proof. We proceed by induction on $|t|$. For type reasons, t is of the form $U\vec{r}$ where U is either a variable among $\vec{x}$ or a constant or a symbol $!, \kappa, d_\sigma$, or else U is a ground type β redex $(\lambda x.s)r$, since t is prenormal. If U is some x_i or 0, then $(i), (ii)$ are obvious.

If U is a symbol among $s_0, s_1, p, !, \kappa$, then $\#\vec{r} = 1$ and r_1 has ground type. Using the induction hypothesis on r_1 one can compute $nf(t[\vec{n}/\vec{x}])$ in at most $1 + |r_1|^2 \leq |t|^2$ steps, and (ii) follows from the induction hypothesis on r_1, since $|nf(t[\vec{n}/\vec{x}])| \leq c_U + |nf(r_1[\vec{n}/\vec{x}])| \leq |t| + \max |n_i|$, where c_U is 1 for $U = !, \kappa$, and 2 else.

Suppose that U is d_σ, and that t is $d_\sigma t_1 t_2 t_3 t_4 \vec{r}$. Observe that $n := nf(t_1[\vec{n}/\vec{x}])$ is a numeral, and that $nf(t[\vec{n}/\vec{x}]) = nf(t_j x \vec{r}[\vec{n}, n/\vec{x}, x])$ for some $j \in \{2,3,4\}$ and for some new variable x. By the induction hypothesis, n can be computed in at most $|t_1|^2$ steps, and every term s occuring in the computation satisfies $|s| \leq |t_1| + \max |n_i|$. Since $t_j x \vec{r}$ is an $\mathcal{R}$-free prenormal srna-term of ground type satisfying $|t_j x \vec{r}| < |t|$, the induction hypothesis gives that $nf(t[\vec{n}/\vec{x}])$ can be computed in at most $|t_1|^2 + |t_j x \vec{r}|^2 \leq (|t_1| + |t_j x \vec{r}|)^2 \leq |t|^2$ steps. As for (ii), observe that $|nf(t[\vec{n}/\vec{x}])| \leq |t_j x \vec{r}| + \max(|n_i|, |n|) \leq |t_j x \vec{r}| + |t_1| + \max |n_i| \leq |t| + \max |n_i|$.

Suppose that U is a ground type β redex $(\lambda x.s)r$. Then s has safe type, since t is an srna-term, and $n := nf(r[\vec{n}/\vec{x}])$ is a numeral. By the induction hypothesis, n can be computed in at most $|r|^2$ steps, and every term s occuring in the computation satisfies $|s| \leq |r| + \max |n_i|$. Defining $t' := pnf(s\vec{r})$, observe that t is a ground type srna-term. Lemma 8 gives that t' can be computed in at most $|s\vec{r}|$ steps, and every term occuring in the computation has a length bounded by $|s\vec{r}|$. In particular, $|t'| \leq |s\vec{r}| < |t|$. So by the induction hypothesis one can compute $nf(t[\vec{n}/\vec{x}]) = nf(t'[\vec{n}, n/\vec{x}, x])$ in at most $|r|^2 + |s\vec{r}| + |s\vec{r}|^2 \leq (1 + |s| + |s\vec{r}|)^2 \leq |t|^2$ steps. As for (ii), observe $|nf(t[\vec{n}/\vec{x}])| \leq |t'| + \max(|n_i|, |n|) \leq |t'| + |r| + \max |n_i| \leq |t| + \max |n_i|$. □

Now lemma 8 and lemma 9 imply the corollary below.

Corollary 2 *For any $\mathcal{R}$-free closed srna-term t of ground type, $nf(t)$ can be computed in at most $|t| + |t|^2$ steps, and every term s occurring in the computation satisfies $|s| \leq |t|$.* □

For computing the normal form of $\mathcal{R}$-free closed srna-terms of ground type, it was enough to consider $\mathcal{R}$-free srna-terms t of ground type and numerals $\vec{n}$ such that $t[\vec{n}/\vec{x}]$ is closed. In order to compute $\mathcal{R}$-free srna-terms t of ground types, we slightly generalise this technique, where the strategy is that we do as much as necessary, but as little as possible.

Definition 7 *For srna-terms r, an* argument for r *is a list $\vec{n}; \vec{x} := n_1, \ldots, n_k; x_1, \ldots, x_k$ where each n_i is a ground type $\mathcal{R}$-free srna-term of the same type as the variable x_i such that n_i is a numeral if it has complete type, and $\forall y \in FV(r).\, y \in \vec{x} \vee y$ is safe.*

Lemma 10 ($\mathcal{R}$-Elimination) *For every safe or ground srna-term t one can find a polynomial q_t such that for every argument $\vec{n}; \vec{x}$ for t there is an $\mathcal{R}$-free srna-term $rf(t[\vec{n}/\vec{x}])$ satisfying*

(i) $rf(t[\vec{n}/\vec{x}])$ can be computed in at most $q_t(\sum |n_i|)$ steps,

(ii) every term s occurring in the computation satisfies $|s| \leq q_t(\sum |n_i|)$,

(iii) $t[\vec{n}/\vec{x}] \longrightarrow^* rf(t[\vec{n}/\vec{x}])$.

Proof. We proceed by induction on $|t|$, presupposing a fixed but arbitrary argument $\vec{n}; \vec{x}$ for t. Throughout the proof let $m := \sum |n_i|$.

Suppose that t is $\lambda x.r$. Then $\vec{n};\vec{x}$ is an argument for r. Using the induction hypothesis, we define $q_t := |t| + q_r$ and $rf(t[\vec{n}/\vec{x}]) := \lambda x.rf(r[\vec{n}/\vec{x}])$. Here $(i), (ii), (iii)$ are obvious.

If t is $U\vec{r}$ where U is either a variable or a symbol $0, s_0, s_1, p, !, \kappa, d$, then $\vec{n};\vec{x}$ is an argument for all $\vec{r}$. Using the induction hypothesis, we define $q_t := |t| + \sum q_{r_i}$ and $rf(t[\vec{n}/\vec{x}]) := U\,rf(\vec{r}[\vec{n}/\vec{x}])$. The properties $(i), (ii), (iii)$ are obvious.

Suppose that t is $(\lambda x.s)r\vec{r}$. We distinguish two cases on the form of t. If $|FO(x,s)| \leq 1$, then $\vec{n};\vec{x}$ is an argument for $t' := s[r/x]\vec{r}$, and $|t'| < |t|$. Using the induction hypothesis, we define

$$q_t := |t| + q_{t'} \text{ and } rf(t[\vec{n}/\vec{x}]) := rf(t'[\vec{n}/\vec{x}]).$$

Otherwise $|FO(x,s)| \geq 2$ and x is ground, and s is safe, since t is an srna-term. Since $\vec{n};\vec{x}$ is an argument for r, let $n := rf(r[\vec{n}/\vec{x}])$ if x is safe, and $n := nf(rf(r[\vec{n}/\vec{x}]))$ otherwise. Observe that if n is complete, then n is a numeral, because then $r[\vec{n}/\vec{x}]$ is closed, since $\vec{n};\vec{x}$ is an argument for r. Using the induction hypothesis and corollary 2, one can compute n in at most $2 \cdot q_r(m) + q_r^2(m)$ steps, and every term occuring in the computation has a length bounded by $q_r(m)$. Defining $t' := s\vec{r}$, observe that t' is a safe srna-term satisfying $|t'| < |t|$, and $\vec{n}, n; \vec{x}, x$ is an argument for t'. Therefore, using the induction hypothesis, we define

$$q_t := |t| + 2 \cdot q_r + q_r^2 + q_{t'} \circ (\mathrm{id} + q_r) \text{ and } rf(t[\vec{n}/\vec{x}]) := rf(t'[\vec{n}, n/\vec{x}, x]).$$

Finally, suppose that U is $\mathcal{R}_\sigma$, and that t is $\mathcal{R}_\sigma ghn\vec{r}$. Now observe that $h = !(\lambda x^{!\iota}.H)$ for some safe H, and $\vec{n};\vec{x}$ is an argument for all $g, n, \vec{r}$. Furthermore, for every numeral k of type $!\iota$ the extended list $\vec{n}, k; \vec{x}, x$ is an argument for H. Let $l := \#\vec{r}$. Applying the induction hypothesis to all $g, H, n, \vec{r}$, we define

$$\begin{aligned} q_t &:= |t| + 2 \cdot q_n + q_n^2 + q_n \cdot (1 + Q) + q_g + l + \sum q_{r_i} \\ Q &:= q_n + q_H \circ (\mathrm{id} + q_n) \\ rf(t[\vec{n}/\vec{x}]) &:= (T_0(\ldots(T_{|N|-1}G)\ldots))R_1 \ldots R_l \end{aligned}$$

where $!N := nf(rf(n[\vec{n}/\vec{x}]))$, $T_i := rf(H[\vec{n}, k_i/\vec{x}, x])$ with $k_i := nf(!\,p^{(i)}N)$, $G := rf(g[\vec{n}/\vec{x}])$, and $R_j := rf(r_j[\vec{n}/\vec{x}])$ for $j = 1, \ldots, l$. Since n has complete type, $n[\vec{n}/\vec{x}]$ is closed and ground. As above it follows that $!N$ is a numeral computable in at most $2 \cdot q_n(m) + q_n^2(m)$ steps, and every term occurring in the computation has a length bounded by $q_n(m)$. Having computed N, every k_i can be computed in $i \leq q_n(m)$ steps, and every term occurring in the computation has a length bounded by $q_n(m)$. It follows that every T_i is computable in at most $Q(m)$ steps, and $|T_i| \leq Q(m)$. Using the induction hypothesis, $rf(t[\vec{n}/\vec{x}])$ is an $\mathcal{R}$-free srna-term, and $t[\vec{n}/\vec{x}] \longrightarrow^* rf(t[\vec{n}/\vec{x}])$. Part (ii) follows from the induction hypothesis and

$$\begin{aligned} |rf(t[\vec{n}/\vec{x}])| &\leq |N| + (\sum_{i<|N|} |T_i|) + |G| + l + \sum |R_j| \\ &\leq q_n(m) \cdot (1 + Q(m)) + q_g(m) + l + \sum q_{r_j}(m) \\ &\leq q_t(m). \end{aligned}$$

As for (i), in a similar way one just has to add up all steps needed to compute $rf(t[\vec{n}/\vec{x}])$. □

Up to now we have focussed on the number of steps needed to compute the normal form of an srna-term. Now we bring in the time structure by giving a *weight* on each reduction step.

Lemma 11 (Time) *There is a polynomial q_{time} such that for every srna-term t, finding and converting any redex in t, and producing the resulting reduct takes* time *at most $q_{time}(|t|)$.*

What "time" really is and what this polynomial actually looks like depends on the computation model we choose, for example "time" with respect to the Turing machine. However, there is such a polynomial, and that serves our purpose.

Lemma 12 (Bounding) *Let t be any closed rna-term of number theoretical type $\vec{\tau} \to \tau$, that is, all $\vec{\tau}, \tau$ are of ground type. Then there is a polynomial q_t such that for every list of suitable numerals $\vec{n}$ one can compute the numeral $nf(t\vec{n})$ in time $q_t(\sum |n_i|)$.*

Proof. Let $t' := nf(t\vec{x})$ for some new ground type variables $\vec{x}$. Then let $t'' := \alpha(t')$ be the srna-representation of t' obtained from lemma 7, and let c be the time needed to compute t''. Now let $q_{t''}$ be the polynomial for t'' obtained from lemma 10. Then we define

$$q_t(x) := c + (2 \cdot q_{t''}(x) + q^2_{t''}(x)) \cdot q_{time}(q_{t''}(x)).$$

Let $\vec{n}$ be any suitable list of numerals. Then $\vec{n}; \vec{x}$ is an argument for t'' such that $t''[\vec{n}/\vec{x}]$ is closed and ground. Thus, lemma 10 and corollary 2 imply that the numeral $nf(rf(t''[\vec{n}/\vec{x}])) = nf(t\vec{n})$ can be computed in at most $2 \cdot q_{t''}(\sum |n_i|) + q^2_{t''}(\sum |n_i|)$ steps, and every term s occurring in the computation satisfies $|s| \leq q_{t''}(\sum |n_i|)$. Therefore lemma 11 gives that $nf(t\vec{n})$ can be computed in time $q_t(\sum |n_i|)$. □

Corollary 3 (Polytime) *Every closed rna-term of number theoretical type denotes a polynomial time computable function.*

Proof. Let t by a closed rna-term of number theoretical type $\vec{\tau} \to \tau$. Let $f := [\![t]\!]$, and let q_t be the polynomial obtained from lemma 12. Given inputs $\vec{n}$ for f, we construct suitable numerals $\vec{n}'$ of type $\vec{\tau}$, representing $\vec{n}$, such that $|n'_i| = c_i + |n_i|$ for some fixed constant c_i. By lemma 12 one can compute $nf(t\vec{n}')$ in time $q_t(\sum |n'_i|)$. Since $f(\vec{n}) = [\![t\vec{n}']\!] = [\![nf(t\vec{n}')]\!]$, we have thus computed $f(\vec{n})$ in time $p_t(|\vec{n}|) := q_t(\sum(c_i + |n_i|))$. □

7 Embedding the polynomial time computable functions

In this last section we complete the proof that the number theoretical functions definable in the system of rna-terms are just the polynomial time computable functions. It remains to embed the polynomial time computable functions into the system of rna-terms. We refer to one of the resource-free function algebra characterisations of the polynomial time computable functions as given in [1] or in [2] or else in [5].

Lemma 13 *Every polynomial time computable function f has a representation t_f in the system of rna-terms.*

Proof. In [1] the polynomial time computable functions are characterised by a function algebra B based on the schemata of safe recursion and safe composition. There every function is written in the form $f(\vec{x};\vec{y})$ where $\vec{x};\vec{y}$ denotes a kind of bookkeeping of those variables $\vec{x}$ involved in a safe recursion in the definition of f, whereas $\vec{y}$ denotes those variables on which no recursion has been performed. We proceed by induction on the definition of $f(\vec{x};\vec{y})$ in B, associating to f a closed rna-term t_f of type $!\vec{\iota},\vec{\iota}\to\iota$ such that t_f is denoting f, i.e. $[\![t_f]\!]=f$.

If f is an initial function $0, s_i(;y), p(;y)$, then $t_f := f$. If f is a projection $\pi^{m,n}_j$ satisfying $\pi^{m,n}_j(x_1,\ldots,x_n;x_{n+1},\ldots,x_{m+n}) = x_j$, then we define $t_f := \lambda x_1\ldots x_{m+n}.u_j$ where $u_j := \kappa x_j$ if $j \le m$, and $u_j := x_j$ otherwise. If f is the conditional c satisfying $c(;y_1,y_2,y_3) = y_2$ if $y_1 mod\ 2 = 0$, and $c(;y_1,y_2,y_3) = y_3$ otherwise, then $t_f := \lambda y_1y_2y_3.d_\iota y_1(\lambda z.y_2)(\lambda z.y_2)(\lambda z.y_3)$ where all $\vec{y},z$ are of type ι.

If f is defined by safe composition from $g,\vec{g},\vec{h}$, that is, $f(\vec{x};\vec{y}) := g(\vec{g}(\vec{x};);\vec{h}(\vec{x};\vec{y}))$ where $\#\vec{g} =: m$ and $\#\vec{h} =: n$, then using the induction hypothesis we define

$$t_f := \lambda\vec{x}^{!\vec{\iota}}\vec{y}^{\vec{\iota}}.t_g(!\,t_{g_1}\vec{x})\ldots(!\,t_{g_m}\vec{x})(t_{h_1}\vec{x}\vec{y})\ldots(t_{h_n}\vec{x}\vec{y}).$$

Finally, if f is defined by safe recursion from g,h, that is, $f(0,\vec{x};\vec{y}) = g(\vec{x};\vec{y})$ and $f(s_ix,\vec{x};\vec{y}) = h(s_ix,\vec{x};f(x,\vec{x};\vec{y}),\vec{y})$ for $s_ix \neq 0$, then using the induction hypothesis we define

$$t_f := \lambda x^{!\iota}x_1^{!\iota}\ldots x_m^{!\iota}.\mathcal{R}_\sigma(t_g\vec{x})\,!(\lambda u^{!\iota}v^{\iota}.t_hu\vec{x}v)x$$

where $\sigma := \vec{\iota}\to\iota$ and $\#\vec{x} =: m$. In each case is easy to verify that t_f is denoting f. □

Combining corollary 3 and the previous lemma, we obtain that the expressive power of rna-terms is just the polynomial time computable functions.

Theorem 4 *A number theoretical function is polynomial time computable if and only if it is definable in the system of rna-terms.* □

References

[1] S. J. Bellantoni and S. Cook. A New Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity*, 2:97–110, 1992.

[2] S. J. Bellantoni and K.-H. Niggl. Ranking Primitive Recursions: The Low Grzegorczyk Classes Revisited. *SIAM Journal of Computing*, 1998. To appear.

[3] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[4] D. Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–190, 1925.

[5] K.-H. Niggl. The μ-Measure as a Tool for Classifying Computational Complexity. Submitted, September 1997.

[6] H. Simmons. The Realm of Primitive Recursion. *Archive for Mathematical Logic*, 27:177–188, 1988.